Implicit Finite Element Technology for Transient Simulation in Electromagnetics

Daniel White

3rd Biennial Tri-Laboratory Engineering Conference on Modeling and Simulation

Nov 2-3, 1999

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract no. W-7405-Eng-48.

UCRL-VG-136070

Implicit Finite Element Technology for Transient Simulation in Electromagnetics

- Goals of the **EM*Solve*** project
  - research the application of vector finite element methods for full-wave computational electromagnetics (CEM)
  - develop a prototype framework for general-purpose CEM
  - solve some interesting application problems in electrostatics, micromagnetics, optics, accelerators, magnetohydrodynamics, etc.
- This presentation
  - Vector finite elements
  - Various formulations of Maxwell's equations
  - Overview of the **EM*Solve*** software architecture
  - Results

Different physical quantities require different discrete representations

- Scalars can be classified as continuous or discontinuous
- Vectors can be classified as having tangential or normal continuity
- Therefore we have four basic types of physical quantities, and we require four different discrete representations

	0-FORM	*1-FORM*	*2-FORM*	*3-FORM*
Integral	Point	Line	Surface	Volume
Derivative	Gradient	Curl	Divergence	None
Continuity	Total	Tangential	Normal	None
Hilbert space	H(grad)	H(curl)	H(div)	L2
E&M	Potential	Fields	Fluxes	Density
Finite element	N(nodal)	W(edge)	F(face)	S(volume)

Vector finite element basis functions are used to discretize vector fields

- Electric field is expanded in terms of the "edge" basis functions W

$$E = \Sigma e_i \, W_i \quad \text{where} \quad e_i = \int E \bullet \hat{t}_i dl$$

  - E is uniquely determined in each cell by the 12 voltages e_i

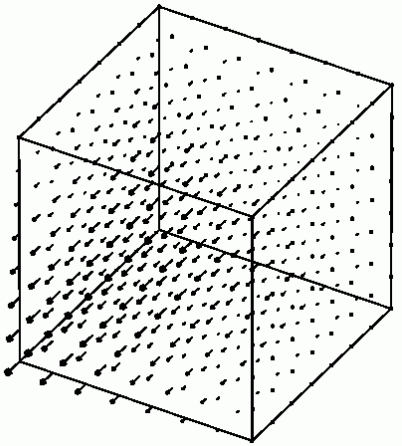
- Magnetic flux density is expanded in terms of the "face" basis functions F

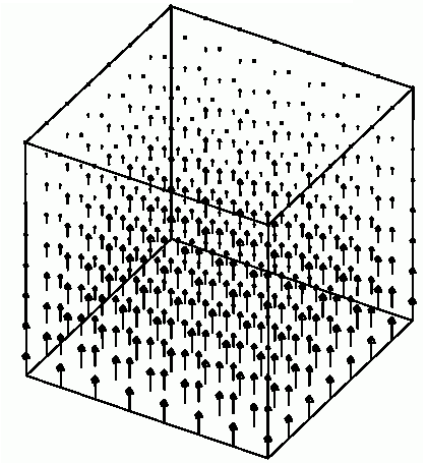
$$B = \Sigma b_i \, F_i \quad \text{where} \quad b_i = \int B \bullet \hat{n}_i da$$

  - B is uniquely determined in each cell by the 6 fluxes b_i

edge element

$$\int W_i \bullet \hat{t}_j dl = \delta_{ij}$$





face element

$$\int F_i \bullet \hat{n}_j da = \delta_{ij}$$

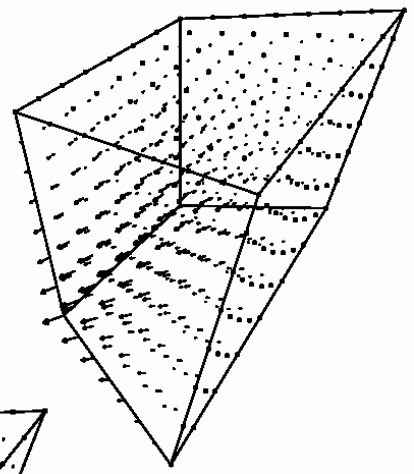
EM*Solve*

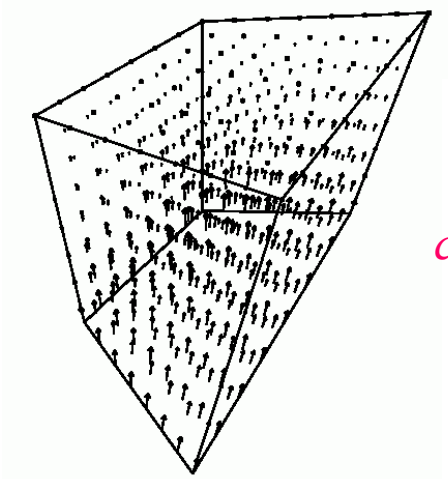
Vector finite elements can be defined on the standard "zoo" elements

- Typically, basis functions are first defined on a reference element and then transformed accordingly
- Covariant transformation preserves line integrals, appropriate for "edge" basis functions
- Contravariant transformation preserves surface integrals, appropriate for "face" basis functions

covariant

$$\tilde{\vec{W}} = J^{-1}\vec{W}$$



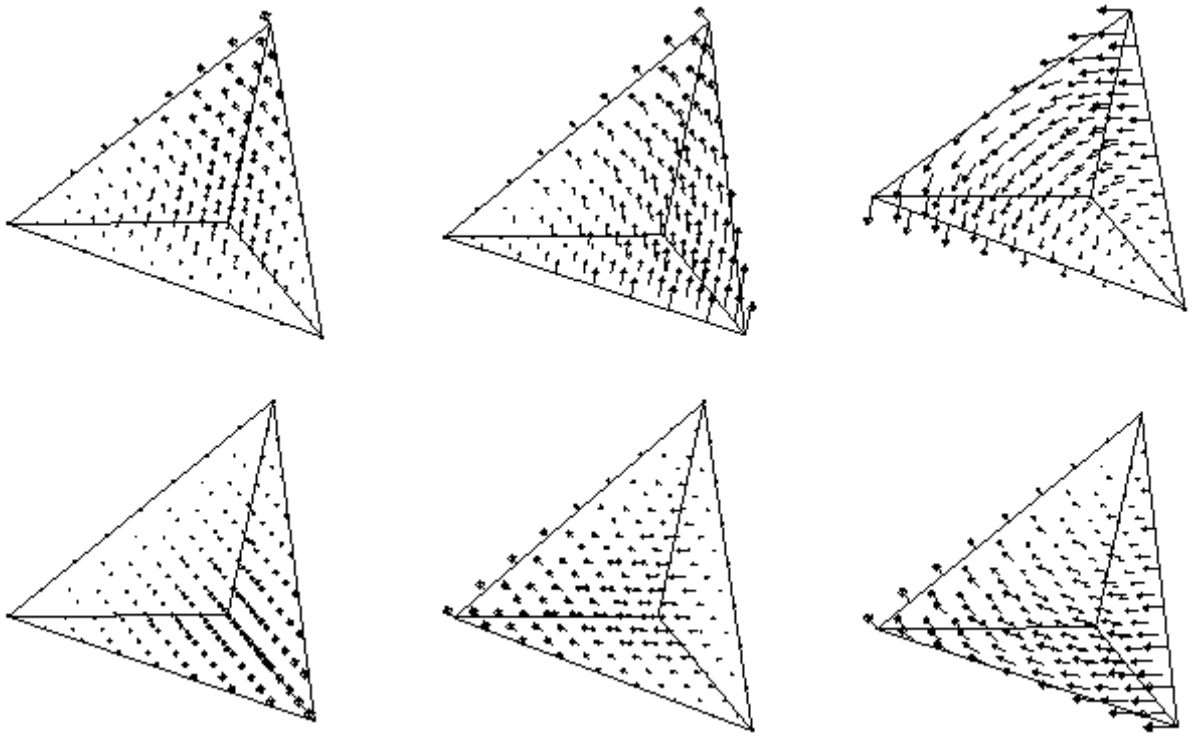


contravariant

$$\tilde{\vec{F}} = J^{T}\vec{F}$$

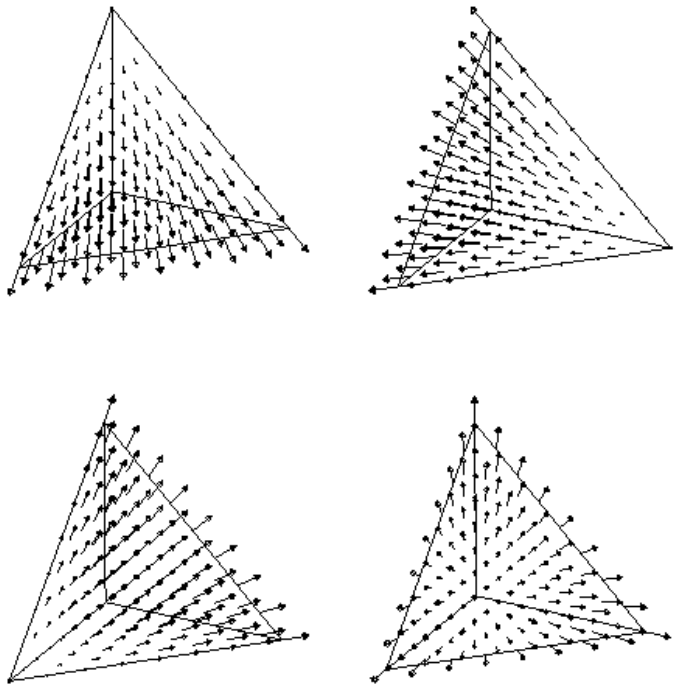
Linear edge functions defined on tetrahedron

- Six basis functions, one for each edge



Linear face functions defined on a tetrahedron

- Four basis functions, one for each face



We can think of differential operators as maps between function spaces

- The operators ∇, $\nabla\times$, and $\nabla\bullet$ are the standard operators
- For example,

  $if\ \phi \subset H(grad)\ then\ \mathrm{E} = \nabla\phi \subset H(curl)$

- The operators $\tilde{\nabla}$, $\tilde{\nabla}\times$ and $\tilde{\nabla}\bullet$ are the weak differential operators (also called adjoint operators)
- For example,

  $if\ E \subset H(curl)\ we\ define\ \ \tilde{\nabla}\bullet E = q$

  $where$

  $\langle q,\phi\rangle = \langle E,\nabla\phi\rangle\ \forall\ \phi \subset H(grad)$

domain

range

	H(grad)	H(curl)	H(div)	L2
H(grad)		$\tilde{\nabla}\bullet$		
H(curl)	∇		$\tilde{\nabla}\times$	
H(div)		$\nabla\times$		$\tilde{\nabla}$
L2			$\nabla\bullet$	

Discrete differential operators are simply sparse matrices

- Matrices depend upon the mesh and the order of the basis functions
- Matrices can easily be combined to form compound discrete operators
- Ideal abstraction for parallel CEM

domain

range	node	edge	face	cell
node		$\mathbf{N}^{-1}\mathbf{P}^{T}\mathbf{A}$		
edge	$\mathbf{P}$		$\mathbf{A}^{-1}\mathbf{K}^{T}\mathbf{D}$	
face		$\mathbf{K}$		$\mathbf{D}^{-1}\mathbf{Q}^{T}\mathbf{M}$
cell			$\mathbf{Q}$	

These discrete differential operators have some very nice properties

- Self-adjoint PDE's map to symmetric positive definite matrices
  - required for numerical stability of many time integration methods
  - makes solution of linear systems easier
- Vector identities *strictly* satisfied

$$\nabla \times \nabla \phi = 0 \qquad \nabla \bullet \nabla \times E = 0$$

  - required for strict conservation of physical properties
- For Cartesian meshes, these discrete operators reduce to well known finite-difference formula
  - no need to maintain a separate FD code
  - peace-of-mind

Examples

- Laplace's equation (ϕ approximated as 0-form)

continuous

$$\nabla \bullet \varepsilon \nabla \phi = \rho$$

discrete

$$\underbrace{\mathbf{P}^T \mathbf{A} \mathbf{P}}_{\text{s.p.d}} \phi = \mathbf{N} \rho$$

- Laplace's equation (ϕ approximated as 3-form)

continuous

$$\nabla \bullet \varepsilon \nabla \phi = \rho$$

discrete

$$\mathbf{Q} \underbrace{\mathbf{D}^{-1} \mathbf{Q}^T}_{\text{s.p.d}} \mathbf{M} \phi = \rho$$

Examples (cont.)

- Vector identity $\nabla \times \nabla \phi = 0$

  ϕ approximated as 0-form: $\mathbf{KP}\phi = 0 \ \forall \phi$

  ϕ approximated as 3-form: $\mathbf{A}^{-1}\mathbf{K}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{M}\phi = 0 \ \forall \phi$

- Vector identity $\nabla \bullet \nabla \times E = 0$

  E approximated as 1-form: $\mathbf{QK}e = 0 \ \forall e$

  E approximated as 2-form: $\mathbf{N}^{-1}\mathbf{P}^{\mathrm{T}}\mathbf{K}^{\mathrm{T}}\mathbf{D}e = 0 \ \forall e$

Examples (cont.)

- Vector Helmholtz equation

continuous

$$\nabla \times \mu^{-1} \nabla \times E - \varepsilon \omega^2 E = 0$$

discrete

$$\left(\mathbf{K}^T \mathbf{D} \mathbf{K} - \omega^2 \mathbf{A} \right) e = 0$$

- Time dependent Maxwell's equations

continuous

$$\dot{B} = -\nabla \times E - \sigma_B B - M$$
$$\varepsilon \dot{E} = \nabla \times \mu^{-1} B - \sigma_E E - J$$

discrete

$$\dot{b} = -\mathbf{K} e - \mathbf{G} b - m$$
$$\mathbf{A} \dot{e} = \mathbf{K}^T \mathbf{D} b - \mathbf{R} e - j$$

Example time integration methods

- Explicit leapfrog
  - "Mass lumping", recommended on Cartesian regions only.
  - Equivalent to classic FDTD method.

$$b^{n+1/2} = b^{n-1/2} - \Delta t(\mathbf{K}e^{n} + \mathbf{G}b^{n-1/2} + m^{n})$$
$$e^{n+1} = e^{n} + \Delta t(\mathbf{K}^{T}b^{n+1/2} - \mathbf{R}e^{n} - j^{n})$$

- Semi-Implicit leapfrog
  - Solve self-consist "mass" matrix at every time step. Stability independent of conductivity, improved dispersion characteristics

$$(\mathbf{D} + \tfrac{\Delta t}{2}\mathbf{G})b^{n+1/2} = -\Delta t\mathbf{D}\mathbf{K}e^{n} + (\mathbf{D} - \tfrac{\Delta t}{2}\mathbf{G})b^{n-1/2} - \mathbf{D}m^{n}$$
$$(\mathbf{A} + \tfrac{\Delta t}{2}\mathbf{R})e^{n+1} = \Delta t\mathbf{K}^{T}\mathbf{D}b^{n+1/2} + (\mathbf{A} - \tfrac{\Delta t}{2}\mathbf{R})e^{n} - \mathbf{A}j^{n+1}$$

Example time integration methods (cont.)

- Explicit wave equation

$$e^{n+1} = e^{n-1} + (2\mathbf{I} - \Delta t^2 \mathbf{K}^T \mathbf{D} \mathbf{K}) e^n + j^n$$

- Semi-Implicit wave equation

$$(\mathbf{A} + \tfrac{\Delta t}{2}\mathbf{R}) e^{n+1} = (\mathbf{A} - \tfrac{\Delta t}{2}\mathbf{R}) e^{n-1} + (2\mathbf{I} - \Delta t^2 \mathbf{K}^T \mathbf{D} \mathbf{K}) e^n + \mathbf{A} J^n$$

- Newmark-Theta variably implicit wave equation

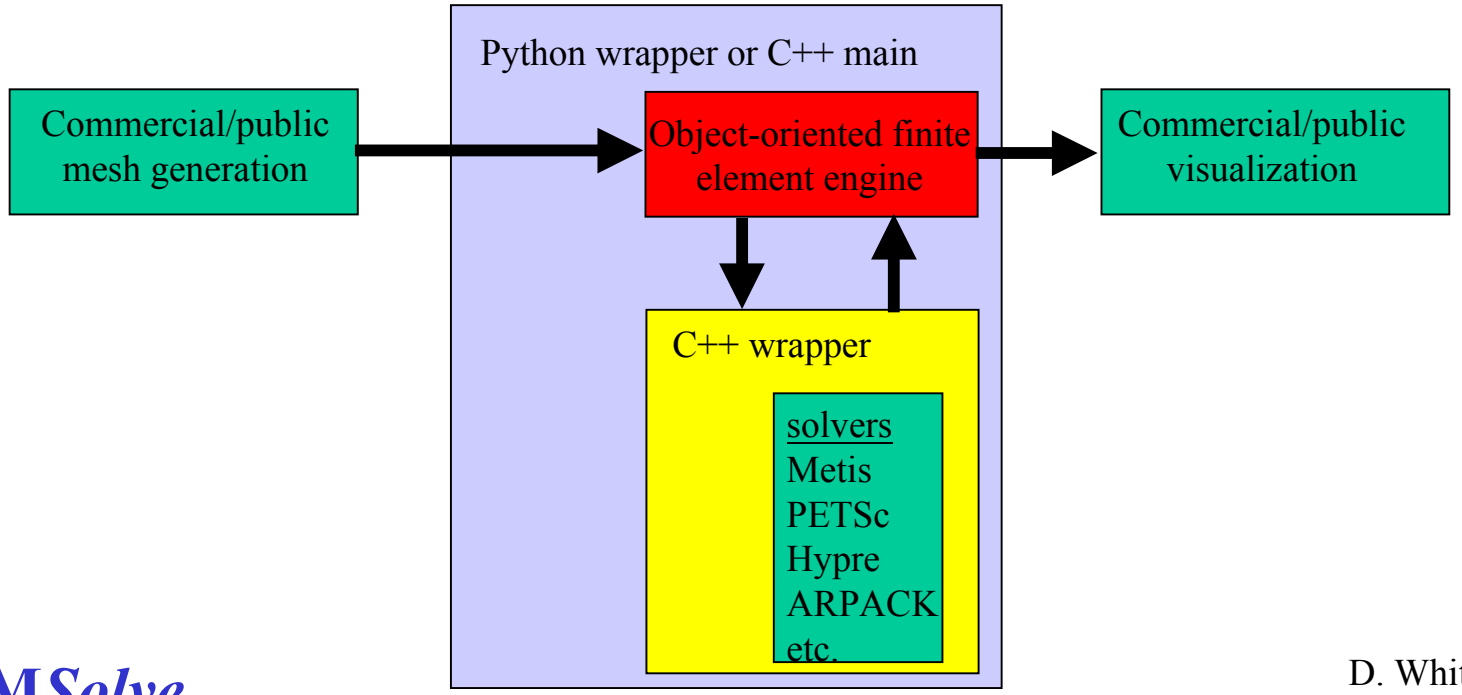
$$(\mathbf{A} + \tfrac{\Delta t}{2}\mathbf{R} + \Delta t^2 \theta \mathbf{K}^T \mathbf{D} \mathbf{K}) e^{n+1} = (\mathbf{A} - \tfrac{\Delta t}{2}\mathbf{R}) e^{n-1} + (2\mathbf{I} - \Delta t^2 (1-\theta) \mathbf{K}^T \mathbf{D} \mathbf{K}) e^n + \mathbf{A} j^n$$

Some notes on “mass” matrices

- Always symmetric positive definite
- On Cartesian grid, Cholesky decomposition of **A** has the exact same sparsity structure as **A**. We can actually put $\mathbf{A}^{-1}$ on r.h.s.
  - D. White, “Solution of Capacitance Systems using Incomplete Cholesky Fixed Point Iteration,” Communications in Numerical Methods in Engineering.
- If an unstructured grid is refined properly, the condition number of **A** is constant. Hence even simple iterative methods are scalable.
  - G. Rodrigue, D. White, J. Koning,”Scalable Preconditioned Conjugate Gradient Inversion of Vector Finite Element Mass Matrices,” Journal of Computational and Applied Mathematics.

It's time to solve the equations...

- We have converted PDE's to algebraic equations.
- Fortunately many good solver packages exist, even for massively parallel computers.



EMSolve currently uses PETSc for all parallel number crunching

- A simple C++ wrapper is employed to make PETSc easier to use, to provide additional error checking, and to enhance extensibility.
- Vector operations
  - $y = y + a*x$, $y = z + a*x$, $y = a*x + b*y$, $a = x^T y$, $a = |x|$, etc.
- Matrix-vector operations
  - $Y = Y + a X$, $y = A*x$, $z = y + A*x$, $a = |A|$, $z = \mathrm{diag}(A)$, etc.
- Linear solvers
  - 10 Krylov methods, 8 preconditioners (and it is easy to add more…)
- Orderings, partitionings, gather and scatter operations.

Example C++ main

- This is the main time stepping loop for semi-implicit leapfrog integration of Maxwell's equations

EM*Solve* overloaded operators use PETSc for automatic parallelization

```
for (int i = 0;i < nstep;i++) {

      // magnetic field update
      sm.set_all(0.0);                    // initialize source vector to zero
      rhs1 = D * sm;                      // add current contribution to rhs
      rhs1 = rhs1 + Q1 * b;               // add Q1 b to rhs, b is the latest magnetic field
      sm = K * e;                         // re-use sm as a temporary intermediate variable
      rhs1 = rhs1 + D * sm;               // D K e is the curl operator, add to rhs
      lsQ.solve(rhs1,b);                  // solve for the new b

      // electric field update
      se.set_all(0.0);                    // initialize source vector to zero
      electricCurrent.set_time(i*dt,i);   // set present simulation time
      electricCurrent.source(se);
      rhs2 = A * se;                      // add current contribution to rhs
      rhs2 = rhs2 + R1 * e;               // add R1 e to rhs, e is the latest electric field
      sm = D * b;                         // re-use sm as a temporary intermediate variable
      rhs2 = rhs2 + KT * sm;              // KT D b is the curl operator, add to rhs
      lsR.solve(rhs2,e);                  // solve for the new e

      // sensing goes here
      snapshot(i,isnap,e,b);               // dump all data if it is time
      sample(i,e);                         // write selected fields to disk
}
```

Example Python script

- The Python interpreter is run on each processor
- Can run interactively, modifying the equations during run-time. Like a massively parallel Matlab with finite element capability

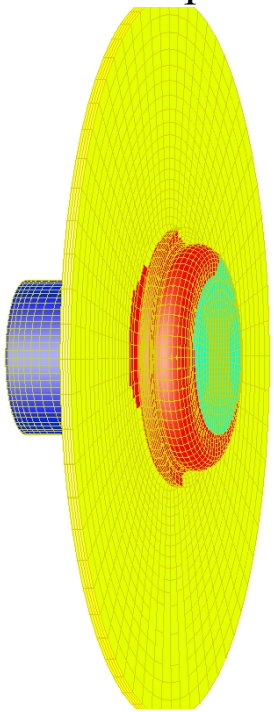
```
for i in range(nstep)
        sm.set_all(0.0)
        rhs1.MatVecMult(G,sm)
        rhs1.MatVecMultPlusVec(Q1,b,rhs1)
        sm.MatVecMult(K,e)
        rhs1.MatVecMultPlusVec(D,sm,rhs1)
        lsQ.solve(rhs1,b)
        se.set_all(0.0);
        electricCurrent.set_time(i,i)
        electricCurrent.source(se)
        rhs2.MatVecMult(A,se)
        rhs2.MatVecMultPlusVec(R1,e,rhs2)
        sm.MatVecMult(D,b)
        rhs2.MatVecMultPlusVec(KT,sm,rhs2)
        lsR.solve(rhs2,e)
```

EM*Solve* Python objects have member functions that use PETSc for automatic parallelization

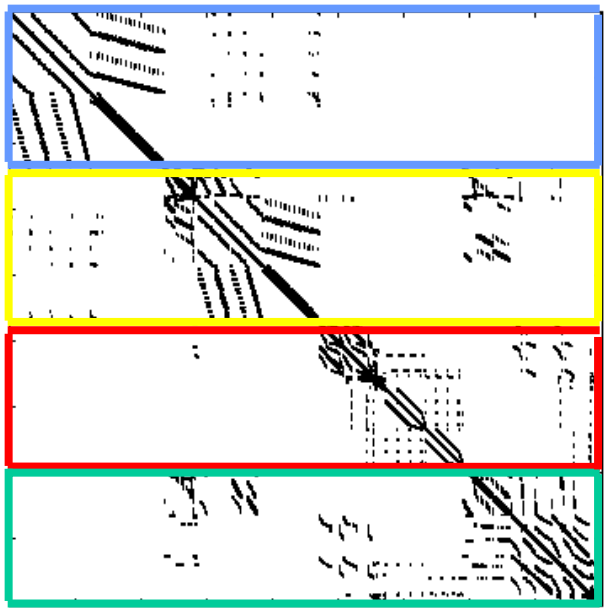
Python is extensible, we can easily add application specific modules as required.

Example four processor decomposition of prototype DARHT cell

Mesh decomposition

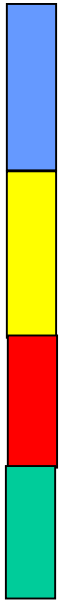


Matrix decomposition



*white-zero, black non-zero
*N = 92,000

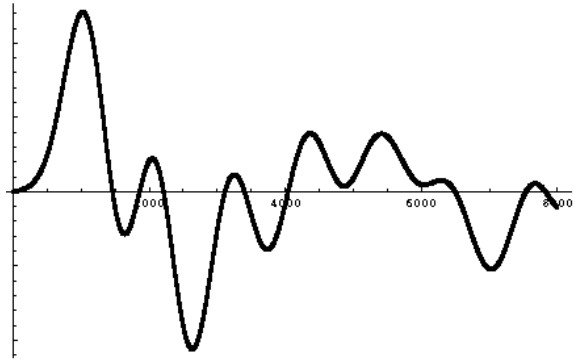
Vector decomposition

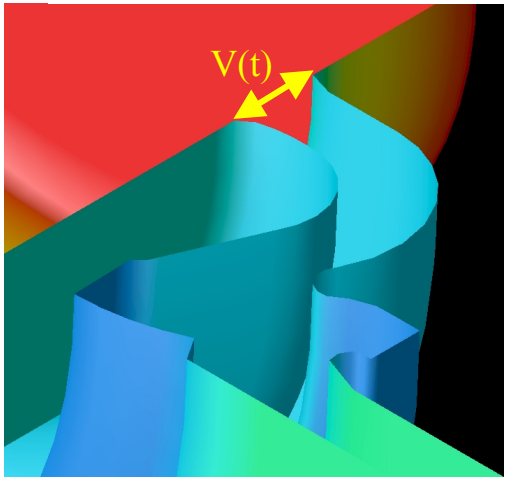


Transient simulation of idealized electron beam in prototype DARHT cell

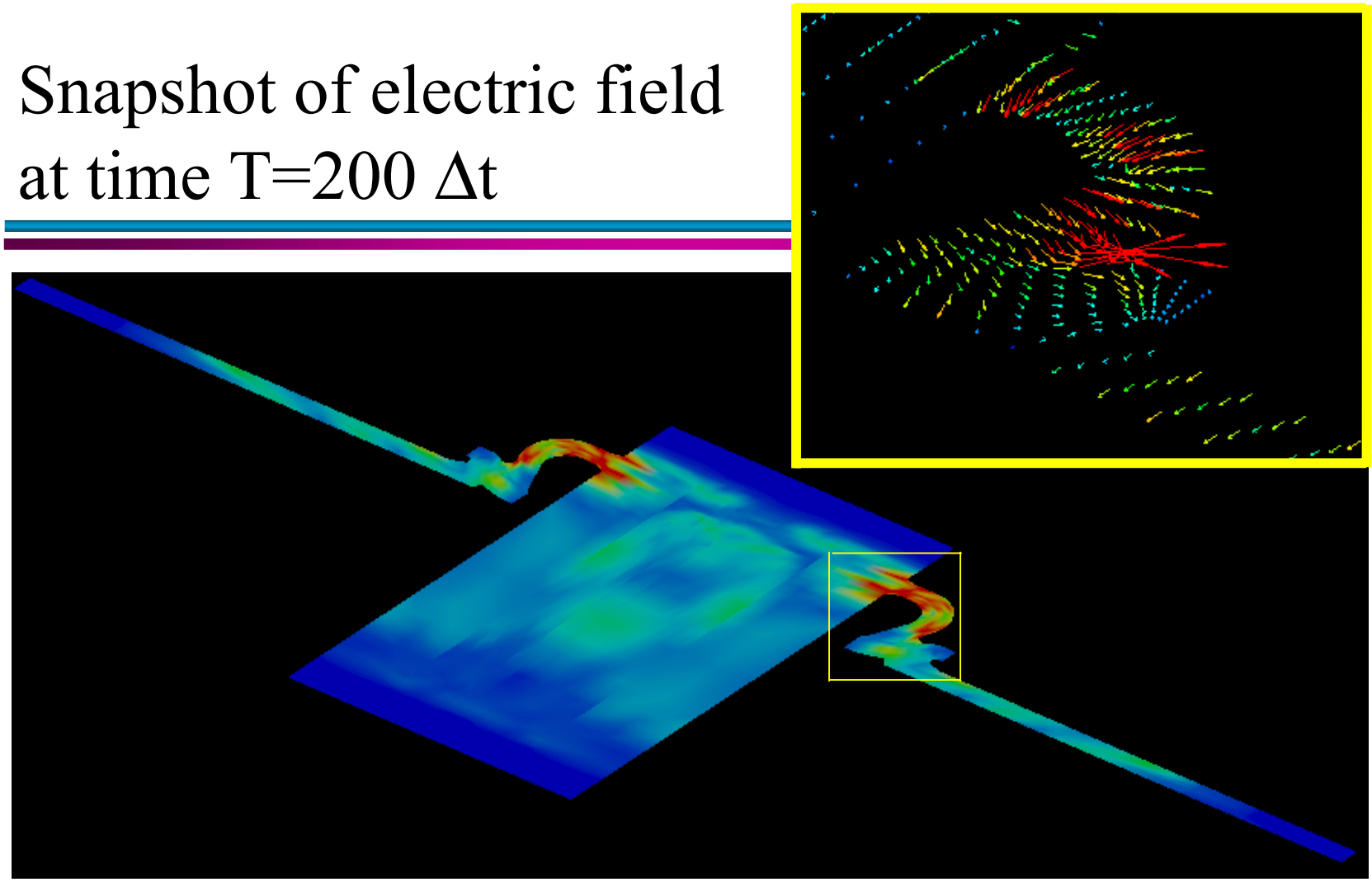
- Input parameters to **EM*Solve***
  - Electric field unknowns: 133645
  - Magnetic field unknowns: 128608
  - Time step: 1.0e-2 (normalized)
  - Time steps: 8000
- Results
  - Average iterations per solve: 20 (Jacobi CG)
  - Total CPU time on 4 processor DEC: 3.7 hours

Longitudinal gap voltage vs time

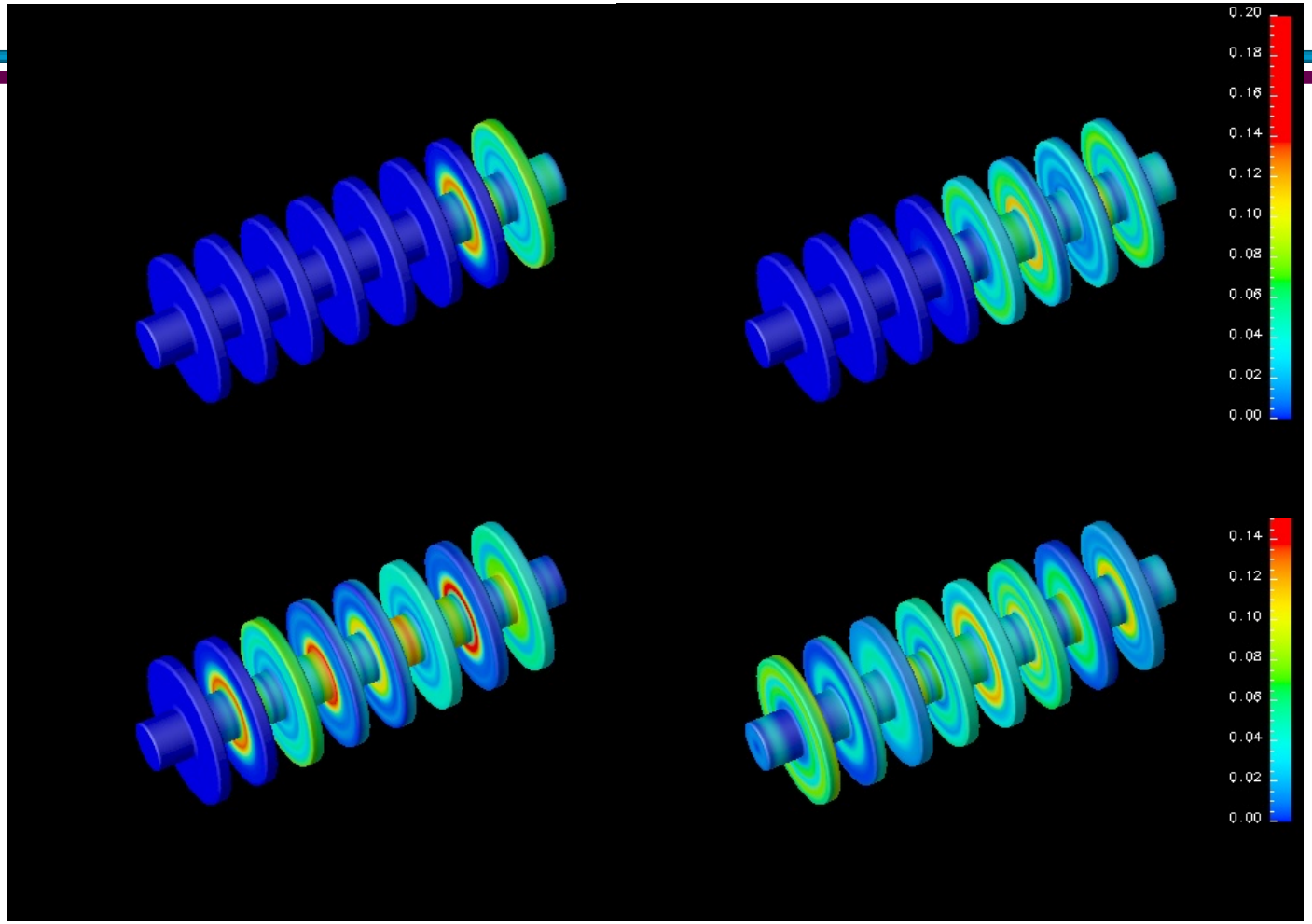




Snapshot of electric field at time T=200 Δt



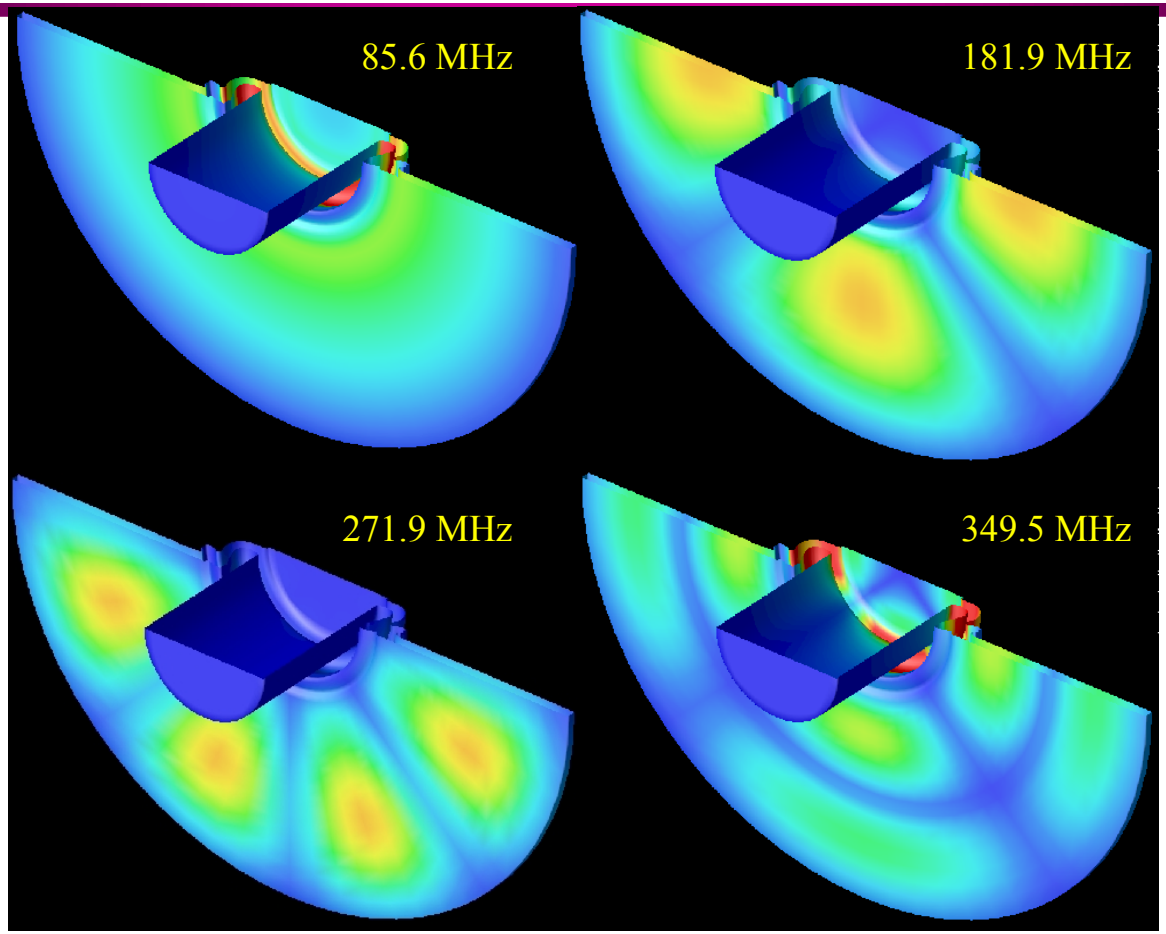
Transient simulation of eight coupled induction cells: 2.4 million unknowns



Prototype DARHT cell eigenmode calculation

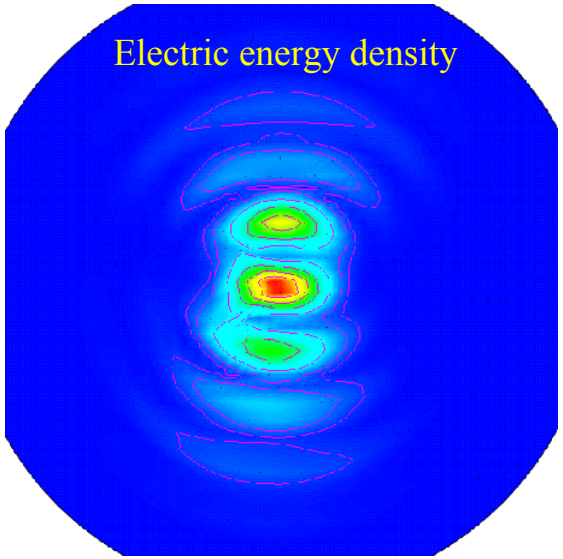
- Input parameters to **EM*Solve***-ARPACK
  - N=92,000
  - 20 eigenvalue-eigenvector pairs
  - 40 Arnoldi vectors
  - eigenvalue tolerance 1.0e-4
  - solve tolerance of 1.0e-7
- Results
  - 5 restarts
  - 82 linear solves
  - 7957 iterations per solve (Jacobi conjugate residual)
  - 15 hours on 4 processor Tera node
  - 3 hours on 32 processors of Blue

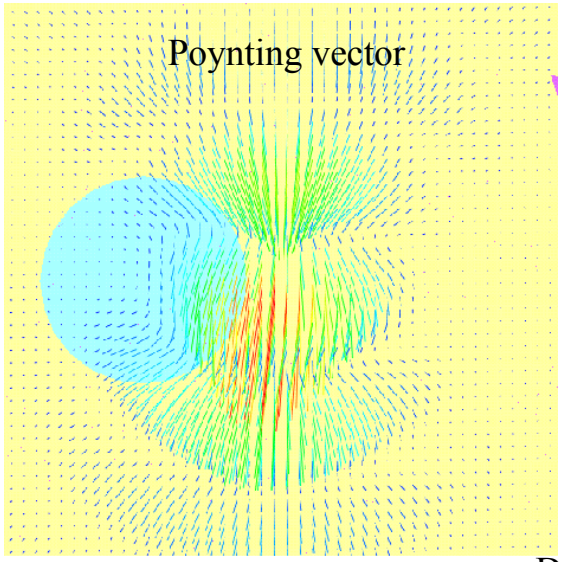
Seleted eigenmodes



An interesting application: optical trapping

- A laser beam is focused towards a dielectric object such that the focus spot is adjacent to the object. The forces on the object are such that the object is pulled towards the focus spot
  - D. White, "Numerical Modeling of Optical Gradient Traps using the Vector Finite Element Method," Journal of Computational Physics.





Things to do...

- Higher order basis functions
  - A big advantage of finite element methods over FD or FV schemes is that it is possible to develop higher order methods for unstructured grids that
    - are provably stable,
    - are provably charge conserving and energy conserving,
    - accurately model discontinuity of fields across interfaces
- Improve efficiency
  - On hybrid grids (structured/unstructured) it is possible to use hybrid time integration methods (explcit/implicit). Up to 30x speedup.
  - Better preconditioners (SPAI, ILUT, multilevel, …). Up to 5x speedup.
- Physics modules
  - Develop as required. Far fields, dispersive/nonlinear media, etc.

Summary

- **EM*Solve*** is a research code used to investigate the utility of vector finite elements. It is not ready for general release.
- The **EM*Solve*** finite element engine uses four different types of finite element basis functions
  - The user has freedom to choose which type of basis function to use for each field, and what type of differential operators are needed.
- The **EM*Solve*** solver uses vectors & matrices as the basic abstractions
  - In the solver, the user has freedom to apply standard algebraic operations with vectors (discrete fields) and matrices (discrete operators)
  - The algebraic operations are automatically parallel
- The **EM*Solve*** numerical method works (stable, conservative, etc.) on unstructured grids, and reduces to standard FD on Cartesian grids.